Hypercube Algorithms on Mesh Connected Multicomputers

Luis Díaz de Cerio, Miguel Valero-García, and Antonio González, *Member*, *IEEE Computer Society*

Abstract—A new methodology named CALMANT (*CC-cube Algorithms on Meshes and Tori*) for mapping a kind of algorithms that we call CC-cube algorithm onto multicomputers with hypercube, mesh, or torus interconnection topology is proposed. This methodology is suitable when the initial problem can be expressed as a set of processes that communicate through a hypercube topology (*a CC-cube* algorithm). There are many important algorithms that fit into the CC-cube type. CALMANT is based on three different techniques: a) the *standard embedding* to assign the processes of the algorithm to the nodes of the mesh multicomputer; b) the *communication pipelining technique* to increase the level of communication parallelism inherent in the CC-cube algorithms; and c) *optimal message-scheduling algorithms* proposed in this work in order to avoid conflicts and minimizing in this way the communication time. Although CALMANT is proposed for multicomputers with different interconnection network topologies, this paper only focuses on the particular case of meshes.

Index Terms—Mapping algorithms, hypercube algorithms, mesh interconnected multicomputers, standard embedding, communication pipelining, message-scheduling algorithms, complete exchange.

1 Introduction

Distributed memory multiprocessors with an interconnection network based on point to point links (multicomputer for short) is a common architecture of parallel computers. Among different interconnection topologies, multidimensional meshes and tori are particularly attractive since they are scalable. Some of the major supercomputer manufacturers have launched multicomputers with a mesh or a torus interconnection network (i.e., the CrayT3E system, Intel Paragon, and Convex SPP).

Designing parallel algorithms for multicomputers is not an easy task. Processes are assigned to the multicomputer nodes and message-scheduling algorithms are defined in order to maximize the processor utilization and minimize communication costs.

In this work, we propose a methodology named CALMANT (Cc-cube ALgorithms on Meshes ANd Tori) for mapping a kind of algorithms that we call *CC-cube* algorithms onto multicomputers with hypercube, mesh, or torus interconnection topology. CALMANT is suitable when the initial problem, after the partition and combination phases, can be expressed as a set of processes that communicate through a hypercube topology (a *CC-cube* algorithm). Although CALMANT is proposed for multicomputers with different interconnection network topologies, this paper only focuses on the particular case of meshes. The initial ideas from which we have developed the CALMANT methodology come from the study of graph embeddings of a hypercube onto a mesh or a torus [5]. This work builds upon a preliminary version of CALMANT for mapping CC-cube algorithms on hypercubes with synchronous communication [2] and asynchronous communication [3]. We are currently extending the proposed methodology to torus interconnected multicomputers.

CALMANT is based on three different techniques: a) the *standard embedding* [10] to assign the processes of the algorithm to the nodes of the mesh multicomputer; b) the *communication pipelining technique* to increase the level of communication parallelism inherent in the CC-cube algorithms; and c) *optimal message-scheduling algorithms* proposed in this work in order to avoid conflicts and minimizing in this way the communication time.

There are many important algorithms that fit into the CC-cube type [5]. Fast Fourier Transform, Complete Exchange communication problem (also known as All-to-All personalized communication), and Jacobi methods for single value decomposition and eigenvalue computation are some examples that can be solved by means of a CC-cube algorithm.

The organization of the rest of the paper is as follows: In Section 2, we present the groundwork on which this work is based. Section 3 describes CALMANT for the case of lines (one-dimensional meshes). The extension to meshes of any dimensionality is described in Section 4. In Section 5, CALMANT is applied to the Complete Exchange communication problem and the results are compared with other proposals particularly designed to solve this problem on meshes. Finally, Section 6 summarizes the main conclusions of this work.

2 Groundwork

This section describes the most important details of the target architectures and the target algorithms we have considered, the standard embedding of hypercubes onto

• *The authors are with the Computer Architecture Department, Universitat Politcnica de Catalunya (UPC), Barcelona, Spain.*
E-mail: {ldiaz, miguel, antonio}@ac.upc.es.

Manuscript received 21 Dec. 1999; revised 14 May 2001; accepted 26 Apr. 2002.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111140.

meshes and the communication pipelining technique. Finally, some notation and definitions are introduced.

2.1 Target Architecture

We assume a distributed memory multicomputer consisting of 2^d nodes connected by point-to-point links in a $(2^{d/c} \times ... \times 2^{d/c})$ c-dimensional mesh topology ($c = 2, 3$). We assume $d/c \geq 2$; otherwise, the architecture is equivalent to a hypercube, which was already studied in [2] and [3].

We assume a synchronous communication protocol, that is, processors are blocked after initiating a communication operation through one of their links, until the communication operation finishes.

Modern parallel computers mostly use a wormhole switching model. As long as there is no congestion, the time for transferring a message of size L between two nodes is almost independent of the distance and can be approximated by:

$$\alpha + L \cdot \tau, \tag{1}$$

where α is the communication start-up and τ is the communication time per size unit. It is also assumed that nodes only have one port available (one-port model), that is to say, nodes can only send and/or receive one message at the same time. Nevertheless, other messages can pass through the node if it is located in the path between the source and the destination nodes of other communication operations.

Finally, we also assume that messages traverse the interconnection network following a dimension-ordering routing policy. In other words, messages use the dimensions of the mesh always following an established order. However, any other routing policy in which messages use the shortest path between the source and destination nodes would not affect the CALMANT methodology, since each message travels along one dimension and, therefore, the shortest path is unique.

2.2 Target Algorithm

The starting point for CALMANT is a CC-cube algorithm. A CC-cube algorithm consist of 2^d processes such that every process communicates with exactly other d processes. These d processes are called its neighbors, and the communication topology of the algorithm is a hypercube. This means that the 2^d processes can be labeled from 0 to $2^d - 1$ in such a way that processes n and n' are neighbors (i.e., they communicate) if the binary codes for n and n' differ in a single bit. If this bit is the ith bit, then n' is the neighbor of n in dimension i and it will be denoted by $n' = N_i(n)$. It is obvious that if $n' = N_i(n)$, then $n = N_i(n')$.

In addition, the code executed by every process of a CC-cube algorithm is the same and it has the following structure:

```
do i=0, K-1
   compute x_i[1 : N] and some local data
   exchange x_i with neighbor in dimension d_i
enddo
```

where d_i is one of the dimensions of the CC-cube ($d_i \in [0, d-1]$).

In the computation stage, N data items are computed. These data are represented by vector $x_i[1:N]$. After a computation stage there is a communication stage in which the computed data x_i are exchanged with one of the neighbors in the CC-cube.

In this paper, we consider only the case of those CC-cube algorithms in which $K = d$ and $d_i \neq d_j$ for all $i \neq j$. The Fast Fourier Transform and the Complete Exchange communication problem are some examples that can be solved by means of a CC-cube with these characteristics. We will assume that $d_i = i$ without loss of generality.

To avoid misunderstandings, from now on CC-cube dimensions will be referred to as just *dimensions* and mesh dimensions will be referred to as *axes*.

2.3 Standard Embedding

Different approaches for embedding hypercube algorithms onto meshes and tori have been proposed [4], [5], [8], [10]. The *standard* embedding [10] has been shown to be optimal for meshes [5] in the sense of reducing the average dilation of the hypercube dimensions.

Let $\langle n_{d-1}, n_{d-2}, ..., n_0 \rangle$ be the binary representation of process n of a CC-cube. Let $\langle m_{c-1}, m_{c-2}, ..., m_0 \rangle$ be the label of node m of a mesh, where the different elements of this label are the coordinates of the node inside the mesh. And let $\langle m_{j,d/c-1}, m_{j,d/c-2}, ..., m_{j,0} \rangle$ be the binary representation of m_j ($j \in [0, c-1]$). Then, the standard embedding of a d-dimensional hypercube onto a mesh, which is denoted by f_{std}, is defined as follows:

$$m = f_{std}(n),$$

where:

$$m_{j,l} = n_{j+l \cdot c}; \forall j \in [0, c-1], \forall l \in [0, d/c-1].$$

The above definition is slightly different to that given by Matic [10] in which $m_{j,l} = n_{(d/c) \cdot j + l}$. It includes a bit permutation in the binary representation of the processes (n). Nevertheless, the minimum average dilation property of the embedding is not affected [5].

Fig. 1 illustrates how the processes of a CC-cube are assigned to the nodes of a mesh when the standard embedding is applied. The shortest paths between any process n and two of its neighbors along two consecutive dimensions ($N_i(n)$ and $N_{i+1}(n)$) follow different axes of the mesh. As it will be shown later, this property, facilitates an efficient scheduling of messages.

Let us define the distance between any two processes n and n' as the number of links in the shortest path between the nodes $f_{std}(n)$ and $f_{std}(n')$ of the mesh. Let i be any of the CC-cube dimensions. The distance between processes n and $N_i(n)$ depends on i but not on n. This property is called *constant dilation property*. Moreover, the shortest path between processes n and $N_i(n)$ uses a single axis of the mesh: the axis $i \bmod c$. We will say that the dimension i of the CC-cube is *mapped* onto the axis $i \bmod c$ of the mesh.

Let $D_i (0 \leq i < d)$ be the dilation of the dimension i of a CC-cube when it is embedded on a mesh through the standard embedding. We can express the value of D_i as:

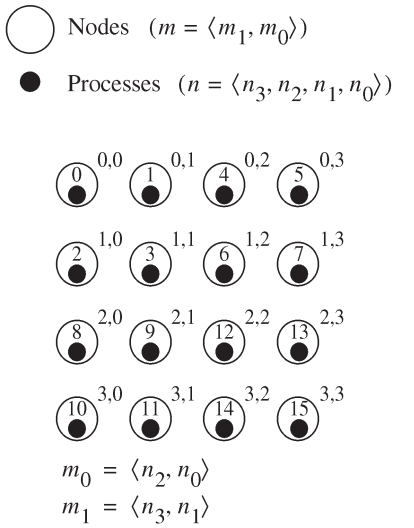

Fig. 1. The standard embedding: a 4-dimensional CC-cube onto a (4 x 4) mesh.

$$D_i = 2^{i/c}. \tag{2}$$

The following list summarizes the main properties of the standard embedding:

- Constant dilation.
- Minimal average dilation.
- The shortest path between any two CC-cube neighbor processes along a given dimension uses a single mesh axis, and this axis only depends on the CC-cube dimension.

When we execute the iteration i of the CC-cube on the mesh, due to the standard embedding properties, all the processes send a message through the same mesh axis to their corresponding neighbors that are located at a distance D_i. Since only one message can traverse a link at the same time in each direction, the cost of the communication stage in each iteration of the CC-cube is proportional to D_i and can be expressed as:

$$t_i = D_i(\alpha + N \cdot \tau),$$

where N is the size of vector x_i.

The computation in one iteration of a CC-cube algorithm requires the data received in the previous one. Since only one CC-cube dimension is used in every iteration, the total time that the algorithm spends in communication is:

$$t = \sum_{i=0}^{d-1} t_i = \sum_{i=0}^{d-1} 2^{i/c}(\alpha + N \cdot \tau) = c(2^{d/c} - 1)(\alpha + N \cdot \tau).$$

The use of only one CC-cube dimension in every iteration prevents an effective exploitation of the communication bandwidth of the mesh. To reduce the communication costs, we will reorganize the CC-cube algorithm so that several messages can be sent in parallel through several or even all the dimensions in every iteration. To introduce this parallelism in the communications, we will apply the communication pipelining technique to the original CC-cube algorithm.

2.4 Communication Pipelining

The *communication pipelining technique* is inspired in the software pipelining approach used to generate code for VLIW processors [9]. It is based on the fact that the whole vector x_{i-1} from neighbor in dimension $i-1$ may not be needed to compute $x_i[j]$. Communication pipelining can be applied to a CC-cube in those cases in which the computation of a given element $x_i[j]$ can be expressed as:

$$x_i[j] = f(x_{i-1}[1:j], x_{i-2}[1:j+1], \ldots, x_1[1:j+i-2], \text{local data}).$$

In other words, $x_i[j]$ must be a function of the j first elements received through dimension $i-1$, and the $j+1$ first elements received through dimension $i-2$, and so on.

Communication pipelining is based on splitting the vector x_i into Q packets of size N/Q[1] and rewriting the algorithm as follows: In the first iteration, every node computes the first packet of x_0 and sends the result to its neighbor in dimension 0. In the second iteration, every node computes the second packet of x_0 and the first packet of x_1. At the end of this second iteration, each node exchanges two messages, one of them with its neighbor in dimension 0, containing the second packet of x_0, and the other one with its neighbor in dimension 1, containing the first packet of x_1. Both packets can be sent in parallel. Proceeding in this way, at the end of the third iteration, every node can send three messages in parallel. Following this approach, a parallel algorithm that makes use of all the dimensions of the CC-cube at the same time can be designed. The resulting algorithm is referred to as *pipelined CC-cube*. Fig. 2 shows how a 3-dimensional CC-cube is transformed when the communication pipelining technique is applied. Data that is computed by the processes in every iteration of the algorithms are represented by boxes. Different gray levels identify data belonging to different iterations of the original CC-cube. Moreover, the links through which communication takes place in every iteration are highlighted.

The value of Q is referred to as the *pipelining degree*. A pipelined CC-cube algorithm consists of three different phases: *prologue*, *kernel*, and *epilogue*. If $Q < d$, the prologue and the epilogue are composed of $Q-1$ iterations each and the kernel consists of $d-Q+1$ iterations. The communication in the kth iteration of the prologue involves the dimensions 0 to $k-1$ of the CC-cube. The kth iteration of the kernel uses the dimensions $k-1$ to $Q+k-2$ and, finally, the kth iteration of the epilogue involves the dimensions $d-Q+k$ to $d-1$. We refer to this situation as *shallow* pipelining because all d dimensions of the CC-cube are never used simultaneously. If $Q \geq d$, the prologue and the epilogue are composed of $d-1$ iterations each and the kernel consists of $Q-d+1$ iterations. In this case, the kth iteration of the prologue uses the dimensions 0 to $k-1$. All iterations of

1. For the sake of clarity, we assume that N is a multiple of Q. It can be easily shown that the analytical models developed in this paper introduce a negligible error when they are used for arbitrary values of N and Q.

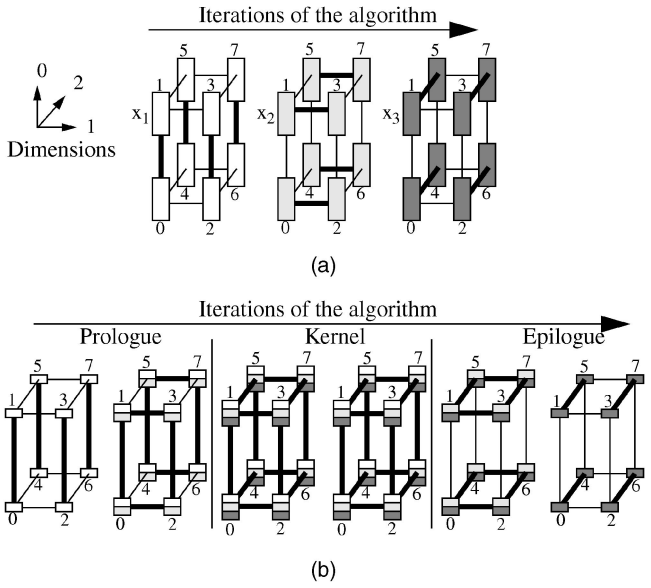

Fig. 2. The communication pipelining technique: (a) a 3-dimensional CC-cube without communication pipelining and (b) a 3-dimensional CC-cube with communication pipelining.

the kernel involve the dimensions 0 to $d-1$ and, finally, the kth iteration of the epilogue communicates through the dimensions k to $d-1$. We refer to this situation as *deep* pipelining because every iteration of the kernel uses all the dimensions of the CC-cube simultaneously. Fig. 2b is a case of deep pipelining since $d=3$ and $Q=4$.

The communication pipelining technique introduces a certain level of communication parallelism in the pipelined CC-cube that depends on the value of Q. An increment in Q increases the parallelism in the communication but results in a greater number of messages to be transmitted. Thus, to minimize the total communication cost, the optimal Q corresponds to the best trade-off between the level of parallelism and the cost due to message startups.

The communication pipelining technique was previously proposed for solving the FFT on the Connection Machine [6]. Since, in that case, the startup time can be neglected, the authors only considered the case of maximum pipelining $Q=N$. Afterwards, we generalized this technique introducing the concept of pipelining degree for mapping CC-cube algorithms on hypercubes with synchronous [2] and asynchronous [3] communications and proposed some approaches to evaluating the optimal pipelining degree.

2.5 Notations and Definitions

Next, sections study and evaluate the execution of a pipelined CC-cube algorithm when it is mapped onto a c-dimensional mesh by means of the standard embedding. Before that, we introduce some notations and definitions.

In every iteration of the pipelined CC-cube several messages, all of them of equal size, are exchanged through a set of consecutive dimensions. In general, the task of exchanging M messages through the dimensions i to $i+M-1$ (one through each dimension) when the pipelined CC-cube is mapped onto a c-dimensional mesh, is denoted by $\langle i, M\rangle_c$. An iteration of the kernel phase of the example represented in Fig. 2 should be denoted by $\langle 0, 3\rangle_c$ and the first iteration of the epilogue phase by $\langle 1, 2\rangle_c$.

The *load* of a link due to a task $\langle i, M\rangle_c$ is defined as the number of messages that traverse the link in each direction when the task is performed using a shortest path routing policy. Due to the standard embedding properties, there is just one way to perform this task, which causes any link to be traversed by the same number of messages in each direction. The *maximum load* of a mesh when performing the task $\langle i, M\rangle_c$ is defined as the load of the link with maximum load and it is denoted by $\lambda_{max}\langle i, M\rangle_c$.

Contention in a communication task is difficult to quantify. In order to simplify the evaluation of the communication time that is involved in a task $\langle i, M\rangle_c$, we assume a *synchronous lockstep communication model*. That is to say, time is viewed as a succession of indivisible parts called steps, such that in each step, several communication operations involving messages of equal size, can be performed in parallel. In this way, a *communication step* is defined as the time required to complete the sending of a message of size L from one node to another of the mesh, when there are no message conflicts. This time is given by the expression (1) in Section 2.1. Several messages can simultaneously travel through the mesh during a communication step, provided that there are no conflicts in the use of the resources (i.e., ports and links). This assumption is not made in order to consider a particular characteristic of the architecture but in order to simplify the description and the evaluation of the algorithms. In fact, it is a conservative assumption because synchronization between steps will not be required to perform the communication tasks. As it is shown later, barriers are required only after the algorithm iterations and the communication overhead of a barrier will be denoted by b.

A lower bound on the number of communication steps required to perform the task $\langle i, M\rangle_c$ is denoted by $LB\langle i, M\rangle_c$ and the number of communication steps used by a given scheduling algorithm to perform this task is denoted by $P\langle i, M\rangle_c$. If $LB\langle i, M\rangle_c = P\langle i, M\rangle_c$ for a given algorithm, we can conclude that this algorithm is optimal from the point of view of communication cost. The communication cost of a task is represented by $T\langle i, M\rangle_c$. Due to the fact that all the messages involved in our scheduling algorithms have the same size, the following relation holds:

$$T\langle i, M\rangle_c = P\langle i, M\rangle_c \cdot (\alpha + L\cdot\tau).$$

In the next two sections of this paper, we propose an efficient solution to perform the tasks $\langle i, M\rangle_c$, first for lines and then for c-dimensional meshes.

3 Pipelined CC-cube on a Line

Since every iteration of the pipelined CC-cube performs a communication through a set of consecutive dimensions, the communication stage of each iteration can be represented by means of task $\langle i, M\rangle_1$. For instance, the communication stage of any iteration of the kernel phase when deep pipelining is applied can be represented by $\langle 0, d\rangle_1$. In this case, all the dimensions of the CC-cube (from 0 to $d-1$) are involved in the communication.

	$s(m)$	$s(m)+1$
$m_i = m_{i+1}$	Send$(i+1)$ Receive(i)	Send (i) Receive$(i+1)$
$m_i \neq m_{i+1}$	Send(i) Receive$(i+1)$	Send$(i+1)$ Receive(i)

Fig. 3. Message-scheduling algorithm for the subtask $\langle i,2\rangle_1$.

3.1 Maximum Load and Lower Bound

In this section, we provide an expression for $\lambda_{max}\langle i,M\rangle_1$ (the maximum load of a line due to a task $\langle i,M\rangle_1$) and use this expression to establish a lower bound on the number of communication steps of task $\langle i,M\rangle_1$, which is denoted by $LB\langle i,M\rangle_1$.

Theorem 1. *The maximum load of a line when a task $\langle i,M\rangle_1$ is performed can be computed as follows:*

$$\lambda_{max}\langle i,M\rangle_1 = \begin{cases} (2^{i+M+1}-2^{i+1})/3, & \text{if } M \text{ is even;} \\ (2^{i+M+1}-2^{i})/3, & \text{if } M \text{ is odd .} \end{cases}$$

Proof. See Appendix. □

Theorem 2. *A lower bound on the number of communication steps required to perform the task $\langle i,M\rangle_1$ on a line ($LB\langle i,M\rangle_1$) can be expressed as:*

$$P\langle i,M\rangle_1 \geq \lambda_{max}\langle i,M\rangle_1 = LB\langle i,M\rangle_1.$$

Proof. Since only one message can traverse a link in a given direction at the same time, it is obvious that $P\langle i,M\rangle_1$ must be greater than or equal to $\lambda_{max}\langle i,M\rangle_1$. On the other hand, nodes cannot send (receive) multiple messages at the same time in a one-port line and therefore, $P\langle i,M\rangle_1$ must also be greater than or equal to M. Thus, we have that:

$$P\langle i,M\rangle_1 \geq \max\{\lambda_{max}\langle i,M\rangle_1, M\},$$

but, according to the expressing given by Theorem 1, it can be proven that:

$$\lambda_{max}\langle i,M\rangle_1 \geq \lambda_{max}\langle 0,M\rangle_1 \geq M.$$

□

3.2 An Optimal Message-Scheduling Algorithm

In this section, we propose a message-scheduling algorithm to perform the task $\langle i,M\rangle_1$ that achieves the lower bound on the number of communication steps. The idea is to decompose the task $\langle i,M\rangle_1$ into a set of smaller subtasks in such a way that the sum of the lower bounds of every subtask is $LB\langle i,M\rangle_1$.

The task $\langle i,M\rangle_1$ can be decomposed into the following set of subtasks:

$$\langle i,M\rangle_1 \equiv \begin{cases} \bigcup_{k=0}^{M/2-1}\langle i+2k,2\rangle_1, & \text{if } M \text{ is even;} \\ \bigcup_{k=1}^{(M-1)/2}\langle i+2k-1,2\rangle_1 \cup \langle i,1\rangle_1, & \text{if } M \text{ is odd.} \end{cases}$$

For instance, the task $\langle 0,6\rangle_1$ will be decomposed into the subtasks $\langle 0,2\rangle_1$, $\langle 2,2\rangle_1$, and $\langle 4,2\rangle_1$.

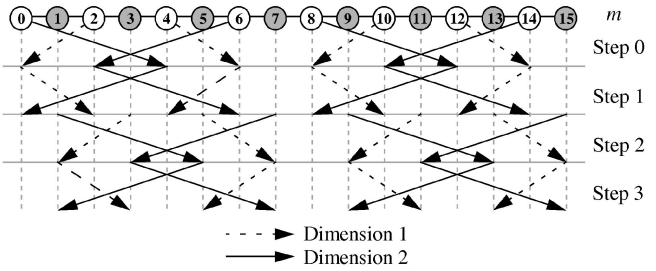

Fig. 4. Communication subtask $\langle 1,2\rangle_1$ on a line of 16 nodes.

Lemma 1. *A lower bound on the number of communication steps to perform the task $\langle i,M\rangle_1$ on a line is given by the following expression:*

$$LB\langle i,M\rangle_1 = \begin{cases} \sum_{k=0}^{M/2-1} LB\langle i+2k,2\rangle_1, & \text{if } M \text{ is even;} \\ LB\langle i,1\rangle_1 + \sum_{k=1}^{(M-1)/2} LB\langle i+2k-1,2\rangle_1, & \text{if } M \text{ is odd.} \end{cases}$$

Proof. It is a direct consequence of Theorem 1 and Theorem 2. □

3.2.1 An Optimal Message-Scheduling Algorithm for the Subtask $\langle i,2\rangle_1$

The nodes of the line are divided into 2^i groups. The group to which a node m belongs is denoted by $g(m)$ and is given by the following expression:

$$g(m) = m \bmod 2^i.$$

Taking into account that nodes can only send and receive one message at a time, a node requires at least two communication steps to exchange messages with its neighbors in dimensions i and $i+1$. In the proposed message-scheduling algorithm, a node m initiates the communication in step $s(m) = 2 \cdot g(m)$ and completes the communication in the next step (that is, it communicates in steps $s(m)$ and $s(m)+1$). Fig. 3 describes a message-scheduling algorithm for exchanging all the messages involved in a group in only two steps. The communication operation to be performed by a node m in each step is determined according to the value of bits i and $i+1$ in the binary representation of m. These bits are denoted by m_i and m_{i+1}, respectively. As an example, nodes such that $m_i = m_{i+1}$ send a message through dimension $i+1$ and receive a message through dimension i during the step $s(m)$. Meanwhile, in this communication step, nodes such that $m_i \neq m_{i+1}$ send a message through dimension i and receive a message through dimension $i+1$. The value of $s(m)$ depends on the group to which node m belongs, and according to the definition of $s(m)$, a different group exchanges the messages every two communication steps.

Fig. 4 shows the case of the subtask $\langle 1,2\rangle_1$ on a line of 16 nodes. Since in this case we have that $i=1$, the nodes are organized into two groups, denoted by white and gray circles. Nodes belonging to the white group communicate during steps 0 and 1, while nodes belonging to the gray group communicate during steps 2 and 3.

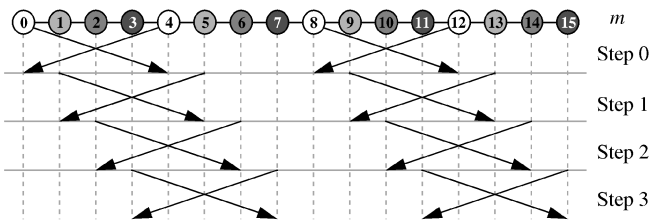

Fig. 5. Communication subtask $\langle 2,\ 1\rangle_1$ on a line of 16 nodes.

The communication through dimension 2 is represented by solid arrows and the communication through dimension 1 is represented by dashed arrows. The communication operations in each group are performed following the algorithm described in Fig. 3. Notice that the proposed message-scheduling algorithm is conflict free because, on one hand, all the messages that travel in the same direction in every communication step do not share any link and, on the other hand, each node sends and receives at most one message at a time, as required in a one port line. This also applies for the general case since the value of i only affects to the communication distance between nodes but not to the communication pattern defined by the algorithm.

Since each group spends two communication steps in completing the communication, the total number of communication steps required to perform the subtask $\langle i,2\rangle_1$ will be twice the number of groups, that is to say:

$$P\langle i,2\rangle_1 = 2^{i+1}.$$

Note that $P\langle i,2\rangle_1 = LB\langle i,2\rangle_1$, and therefore, the proposed approach is optimal.

3.2.2 An Optimal Message-Scheduling Algorithm for the Subtask $\langle i,1\rangle_1$

To describe how the subtask $\langle i,1\rangle_1$ is performed, let us regard the nodes of the line as divided into 2^{i+1} groups. The group to which node m belongs is denoted by $g'(m)$ and is given by the following expression:

$$g'(m) = m \bmod 2^{i+1}.$$

Nodes require one communication step to send a message and receive another through dimension i. The communication step in which the node m sends and receives the corresponding messages is denoted by $s'(m)$. We propose that $s'(m) = g'(m)$, which means that all nodes belonging to the same group send and receive messages during the same communication step. Fig. 5 shows an example for subtask $\langle 2,1\rangle_1$ on a line of 16 nodes. In this case, we have four groups of nodes, represented by different gray levels. Again, it is easy to infer from the figure that the proposed message-scheduling algorithm is conflict free for the general case. Since each group spends one communication step in completing the communication, the total number of communication steps is equivalent to the number of groups:

$$P\langle i,1\rangle_1 = 2^i.$$

Notice that $P\langle i,1\rangle_1 = LB\langle i,1\rangle_1$, and therefore, the proposed approach is optimal.

3.3 Communication Cost

The execution time of the task $\langle i,M\rangle_1$ is given by the number of communication steps multiplied by the cost of every communication step:

$$T\langle i,M\rangle_1 = P\langle i,M\rangle_1 \cdot (\alpha + L\cdot\tau).$$

The expression for $P\langle i,M\rangle_1$ can be evaluated as the sum of the communication steps of every subtask in which task $\langle i,M\rangle_1$ is decomposed:

$$P\langle i,M\rangle_1 = \begin{cases} \sum_{k=0}^{M/2-1} P\langle i+2k,2\rangle_1, & \text{if } M \text{ is even;} \\ P\langle i,1\rangle_1 + \sum_{k=1}^{(M-1)/2} P\langle i+2k-1,2\rangle_1, & \text{if } M \text{ is odd.} \end{cases}$$

From the message-scheduling algorithms described in the previous section, we have that $P\langle i,2\rangle_1 = LB\langle i,2\rangle_1$ and $P\langle i,1\rangle_1 = LB\langle i,1\rangle_1$. According to Lemma 1 and Theorem 2, the previous expression for $P\langle i,M\rangle_1$ can be rewritten as $P\langle i,M\rangle_1 = LB\langle i,M\rangle_1 = \lambda_{max}\langle i,M\rangle_1$ and, therefore, the execution time of the task $\langle i,M\rangle_1$ can be expressed as:

$$T\langle i,M\rangle_1 = \lambda_{max}\langle i,M\rangle_1 \cdot (\alpha + L\cdot\tau).$$

The total communication cost of the algorithm can be evaluated as the sum of the communication cost of every iteration plus the barrier overhead after each iteration. Since the size of every message is $L = N/Q$, for shallow pipelining we have that:

$$T_{shallow}(d,Q) = \sum_{k=0}^{Q-2} T\langle 0,k+1\rangle_1 + \sum_{k=0}^{d-Q} T\langle k,Q\rangle_1 + \sum_{k=0}^{Q-2} T\langle k+d-Q+1,Q-k-1\rangle_1 + (d+Q-1)\cdot b,$$

where b is the cost of a barrier. Likewise, for deep pipelining, we obtain:

$$T_{deep}(d,Q) = \sum_{k=0}^{d-2} T\langle 0,k+1\rangle_1 + (Q-d+1)\cdot T\langle 0,d\rangle_1 + \sum_{k=0}^{d-2} T\langle k+1,d-k-1\rangle_1 + (d+Q-1)\cdot b.$$

These expressions can be used to derive the optimal degree of pipelining (value of Q). The closed formulas for $T_{shallow}$ and T_{deep} can be found in the Appendix.

4 Pipelined CC-cube on a c-dimensional Mesh

This section extends the approach proposed in the previous section to the execution of a pipelined CC-cube on a c-dimensional mesh. The communication stage of each iteration of a pipelined CC-cube on a c-dimensional mesh can be represented by the communication task $\langle i,M\rangle_c$.

4.1 Maximum Load and Lower Bound

Theorem 3. *The maximum load due to the communication task $\langle i, M\rangle_c$ on a $(2^{d/c} \times ... \times 2^{d/c})$ c-dimensional mesh $(\lambda_{max}\langle i, m\rangle_c)$ is equivalent to:*

$$\lambda_{max}\langle \lfloor (i + (M-1) \bmod c)/c \rfloor, \lceil M/c \rceil \rangle_1.$$

Proof. See Appendix. □

Theorem 4. *A lower bound on the number of communication steps required to perform the task $\langle i, M\rangle_c$ on a mesh can be expressed as:*

$$P\langle i, M\rangle_c \geq \max\{\lambda_{max}\langle i, M\rangle_c, M\} = LB\langle i, M\rangle_c.$$

Proof. Since only one message can traverse a link in a given direction at the same time, it is obvious that $P\langle i, M\rangle_c$ must be greater than or equal to $\lambda_{max}\langle i, M\rangle_c$. On the other hand, nodes cannot send (receive) multiple messages at the same time and, therefore, $P\langle i, M\rangle_c$ must also be greater than or equal to M. □

According to Theorem 3 and Theorem 4, we have that:

$$LB\langle i, M\rangle_c = \max\{\lambda_{max}\langle \lfloor (i + (M-1) \bmod c)/c \rfloor, \lceil M/c \rceil \rangle_1, M\}.$$

4.2 A Nearly Optimal Message-Scheduling Algorithm

The task $\langle i, M\rangle_c$ can be decomposed into a set of smaller subtasks in the following way:

$$\langle i, M\rangle_c \equiv \bigcup_{k=1}^{\lfloor M/2c \rfloor} \langle i + M - 2kc, 2c\rangle_c \cup \langle i, M \bmod 2c\rangle_c.$$

For instance, the task $\langle 0, 10\rangle_2$ can be decomposed into the subtasks $\langle 0, 2\rangle_2$, $\langle 2, 4\rangle_2$, and $\langle 6, 4\rangle_2$.

Lemma 2. *The difference between the sum of the lower bounds of the subtasks in the proposed decomposition and $LB\langle i, M\rangle_c$ is under the limits given by the following expression:*

$$\sum_{k=1}^{\lfloor M/2c \rfloor} LB\langle i + M - 2kc, 2c\rangle_c + LB\langle i, M \bmod 2c\rangle_c - LB\langle i, M\rangle_c \leq \begin{cases} 2, & \text{if } c = 2 \text{ and } M \geq 7 \text{ and } i \leq 1; \\ 4, & \text{if } c = 3 \text{ and } M \geq 9 \text{ and } i \leq 4; \\ 0, & \text{otherwise.} \end{cases}$$

Proof. It is an immediate consequence of the expressions given by Theorem 4, Theorem 3, and Theorem 1. □

According to the above lemma, if it exists a solution in which $P\langle i, x\rangle_c = LB\langle i, x\rangle_c$ for any $x \leq 2c$, a nearly optimal algorithm to perform the task $\langle i, M\rangle_c$ can be obtained by sequentially performing every subtask according that solution.

4.2.1 An Optimal Message-Scheduling Algorithm for the Subtask $\langle i, 2c\rangle_c$

Note that the $2c$ dimensions that are involved in the subtask $\langle i, 2c\rangle_c$ are mapped onto the different axes cyclically. In particular, the dimensions k and $k + c$, such that $k \in [i, i + c - 1]$, are mapped onto the axis $k \bmod c$.

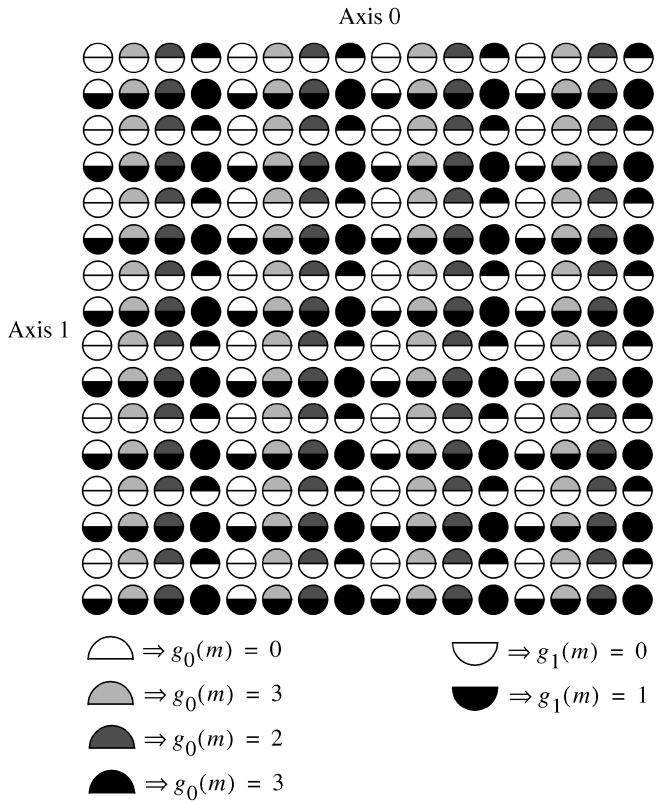

Fig. 6. Group partition for the subtask $\langle 3, 4\rangle_2$ on a (16×16) mesh.

We divide the nodes of the mesh into different groups depending on their position in each of the axes. A node belongs to c different groups. Each group corresponds to one of the axes. If $\langle m_{c-1}, m_{c-2}, ..., m_0\rangle$ is the label of node m, the c groups to which it belongs are given by the following expression:

$$g_{k \bmod c}(m) = m_{k \bmod c} \bmod 2^{\lfloor k/c \rfloor}; \forall k \in [i, i + c - 1].$$

As an example, Fig. 6 shows the case of the subtask $\langle 3, 4\rangle_2$ for a (16×16) mesh. In this example, since $i = 3$ and $c = 2$, we have that $k \in [3, 4]$. The node $\langle m_1, m_0\rangle$ belongs to the groups $g_1(m) = m_1 \bmod 2$ and $g_0(m) = m_0 \bmod 4$ (i.e., node $m = \langle 2, 3\rangle$ belongs to groups $g_1(m) = 0$ and $g_0(m) = 3$). In the figure, the gray level in the upper part of the node representation corresponds to $g_0(m)$ (group corresponding to the horizontal axis) and the lower part corresponds to $g_1(m)$ (group corresponding to the vertical axis).

Note that, for the proposed group division, nodes must communicate only with nodes that belongs exactly to the same groups in every axis of the mesh (nodes that are identically represented in Fig. 6). We consider now the problem of scheduling the messages to be exchanged in every iteration in such a way that conflicts in the mesh are avoided and the communication time is minimized.

When nodes communicate, two restrictions should be taken into account in order to avoid conflicts. First, two nodes belonging to different groups of the same axis should perform the communication through this axis in different communication steps in order to avoid link contention and second, any node should perform the communication

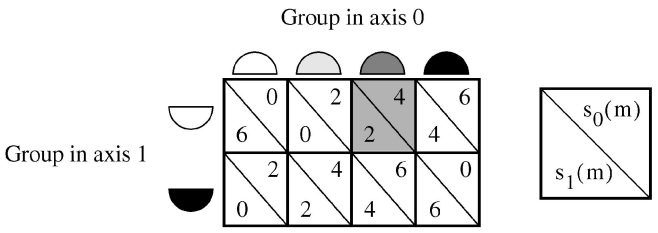

Fig. 7. Scheduling for the solution of the subtask $\langle 3,4\rangle_2$ on a (16×16) mesh.

through different axes of the mesh in different communication steps, since we are assuming a one-port model.

Dimensions k and $k+c$ are the only dimensions of those involved in the subtask that are mapped onto axis $k \bmod c$. Since an axis of a mesh can be considered as a line of $2^{d/c}$ nodes, the communication along the axis can be performed as it was described for lines (in two communication steps). We propose that node m sends and receives the corresponding messages through the dimensions k and $k+c$ during the steps $s_{k \bmod c}(m)$ and $s_{k \bmod c}(m)+1$, where:

$$s_{k \bmod c}(m) = 2\left(\left(\sum_{j=0}^{c-1} g_j(m) + k\right) \bmod \frac{LB\langle i,2c\rangle_c}{2}\right);$$
$$\forall k \in [i, i+c-1]$$

and where the subindex $k \bmod c$ denotes the axis through which the messages are sent. This expression meets the restrictions mentioned before and schedules the activity of the groups using a minimum number of communication steps as it will be shown later. Any other scheduling that satisfies the required restrictions with a minimum number of communication steps would also be an optimal scheduling policy.

Fig. 7 shows the result of applying the proposed scheduling to the example of Fig. 6. The table shows the activity of every node as a function of the group which it belongs to in each axis. The upper parts and lower parts of the boxes represent, respectively, the communication step in which communication through axis 0 and axis 1 is initiated ($s_0(m)$ and $s_1(m)$). As an example, nodes belonging to group 2 in axis 0 and group 0 in axis 1 (shadowed box) communicates along the axis 1 during the communication steps 2 and 3 and communicates along the axis 0 during the communication steps 4 and 5.

Property 1. *The proposed scheduling algorithm is conflict free and its number of communication steps is $P\langle i,2c\rangle_c = LB\langle i,2c\rangle_c$, that is, it is optimal for the subtask $\langle i,2c\rangle_c$.*

Proof. Let us first prove that the scheduling algorithm is conflict free. Let m and m' be two different nodes of the mesh. Let j be an axis of the mesh, such that $m_j \neq m'_j$. Let us assume that $m_k \neq m'_k$ for some $k \neq j$, that is to say, nodes m and m' do not belong to the same line in the mesh. Due to the fact that the communication between two neighbor processes along a given dimension is performed using a single mesh axis, conflicts between messages from/to node m and messages from/to node m' are not possible. Now, let us assume that $m_k = m'_k$ for all $k \neq j$. In this case, $g_k(m) = g_k(m')$. If $g_j(m) \neq g_j(m')$, then $s_j(m) \neq s_j(m')$ and, therefore, conflicts between messages from/to node m and messages from/to node m' are not possible. If $g_j(m) = g_j(m')$, both nodes belongs to the same group of communication and conflicts are avoided using the scheduling algorithm described for lines.

Next, we prove that $P\langle i,2c\rangle_c = LB\langle i,2c\rangle_c$. If x and y are integers and y is an even number we have that:

$$0 \leq 2(x \bmod (y/2)) \leq y-2.$$

Therefore, according to the expression for $s_j(m)$, we obtain:

$$0 \leq s_j(m) \leq LB\langle i,2c\rangle_c - 2.$$

If each group performs the communication in two communication steps, then the last groups perform the communication during steps $LB\langle i,2c\rangle_c - 2$ and $LB\langle i,2c\rangle_c - 1$. The first communication step is denoted by 0 so the total number of communication steps is $LB\langle i,2c\rangle_c$. □

4.2.2 An Optimal Message-Scheduling Algorithm for the Subtask $\langle i,x\rangle_c$ when $1 \leq x \leq c$

Note that the x dimensions that are involved in the subtask $\langle i,x\rangle_c$ are mapped onto the different axes cyclically. In particular, the dimension k ($i \leq k < i+x$) is mapped onto the axis $k \bmod c$.

We divide the nodes of the mesh into different groups. In this case, a node belongs to x different groups. Each group corresponds to one of the axes on which a dimension is mapped. If $\langle m_{c-1}, m_{c-2}, ..., m_0\rangle$ is the label of node m, the x groups to which it belongs are given by the following expression:

$$g_{k \bmod c}(m) = m_{k \bmod c} \bmod 2^{\lfloor k/c \rfloor}; \forall k \in [i, i+x-1].$$

We propose that node m sends and receives the corresponding messages through the dimension k during the step $s_{k \bmod c}(m)$, where:

$$s_{k \bmod c}(m) = \left(\sum_{j=i}^{i+x-1} g_{j \bmod c}(m) + k\right) \bmod LB\langle i,x\rangle_c;$$
$$\forall k \in [i, i+x-1].$$

Among the dimensions that are involved in the subtask, only the dimension k is mapped onto the axis ($k \bmod c$). Since one axis of the mesh can be considered as a line of $2^{d/c}$ nodes, the communication can be performed following the approach described for lines.

Property 2. *The proposed scheduling algorithm is conflict free and its number of communication steps is $P\langle i,x\rangle_c = LB\langle i,x\rangle_c$ so, it is optimal for the subtask $\langle i,x\rangle_c$ when $1 \leq x \leq c$.*

Proof. Similar to Proof for Property 1. □

4.2.3 A Nearly Optimal Message-Scheduling Algorithm for the Subtask $\langle i,x\rangle_c$ when $c < x < 2c$

We divide the nodes of the mesh into different groups. In this case, a node belongs to c different groups. Each group corresponds to one of the axes. If $\langle m_{c-1}, m_{c-2}, ..., m_0\rangle$ is the label of node m, the c groups to which it belongs are given by the following expression:

$$g_{k \bmod c}(m) = \begin{cases} 0, & \text{if} \lfloor k/c \rfloor < 0; \\ m_{k \bmod c} \bmod 2^{\lfloor k/c \rfloor}, & \text{if} \lfloor k/c \rfloor \geq 0; \end{cases}$$
$$\forall k \in [i+x-2c, i+x-c-1].$$

If $k \in [i+x-2c, i-1]$, then $k+c$ is the only dimension among those involved in the subtask that is mapped onto the axis $k \bmod c$. Otherwise, if $k \in [i, i+x-c-1]$, then both dimensions k and $k+c$ are mapped onto the axis $k \bmod c$.

We propose that the node m sends and receives the corresponding messages either through dimensions k and $k+c$ or only $k+c$, during the steps $s_{k \bmod c}(m)$ and $s_{k \bmod c}(m)+1$ using the approach proposed for lines, where:

$$s_{k \bmod c}(m) = 2\left(\left(\sum_{j=0}^{c-1} g_j(m) + k\right) \bmod \left\lceil \frac{LB\langle i,x\rangle_c}{2} \right\rceil\right);$$
$$\forall k \in [i+x-2c, i+x-c-1].$$

Property 3. *The proposed scheduling algorithm is conflict free and the difference between its number of communication steps and $LB\langle i,x\rangle_c$ is:*

$$P\langle i,x\rangle_c - LB\langle i,x\rangle_c = \begin{cases} 1, & \text{if } LB\langle i,x\rangle_c \text{ is odd;} \\ 0, & \text{if } LB\langle i,x\rangle_c \text{ is even.} \end{cases}$$

Proof. Similar to Proof for Property 1. □

This scheduling algorithm is optimal for the subtask $\langle i,x\rangle_c$ when $c < x < 2c$ and $LB\langle i,x\rangle_c$ is even. For $c < x < 2c$, $LB\langle i,x\rangle_c$ is odd only if $x > \lambda_{max}\langle i,x\rangle_c$ and x is odd. This is due to the fact that the expression $x > \lambda_{max}\langle i,x\rangle_c$ always returns an even value.

4.3 Execution Time

The execution time for the task $\langle i,M\rangle_c$ is given by the number of communication steps multiplied by the cost of every communication step:

$$\begin{aligned} T\langle i,M\rangle_c &= P\langle i,M\rangle_c \cdot (\alpha + L\cdot\tau) \\ &= \left(\sum_{k=1}^{\lfloor M/2c \rfloor} P\langle i+M-2kc, 2c\rangle_c \right. \\ &\quad \left. + P\langle i, M \bmod 2c\rangle_c\right) \cdot (\alpha + L\cdot\tau) \\ &= \left(\sum_{k=1}^{\lfloor M/2c \rfloor} LB\langle i+M-2kc, 2c\rangle_c \right. \\ &\quad \left. + LB\langle i, M \bmod 2c\rangle_c + r\right) \cdot (\alpha + L\cdot\tau), \end{aligned}$$

where:

$$r = \begin{cases} 1, & \text{if } M \bmod 2c > \lambda_{max}\langle i, M \bmod 2c\rangle_c \text{ and } M \text{ is even;} \\ 0, & \text{other case.} \end{cases}$$

The total communication cost of the algorithm can be evaluated as the sum of the communication cost of every iteration plus the barrier overhead after each iteration. Since the size of every message is $L = N/Q$, for shallow pipelining we have that:

$$\begin{aligned} T_{shallow}(d,Q) &= \sum_{k=0}^{Q-2} T\langle 0, k+1\rangle_c + \sum_{k=0}^{d-Q} T\langle k, Q\rangle_c \\ &\quad + \sum_{k=0}^{Q-2} T\langle k+d-Q+1, Q-k-1\rangle_c + (d+Q-1)\cdot b, ; \end{aligned}$$

where b is the cost of a barrier. Likewise, for deep pipelining we obtain:

$$\begin{aligned} T_{deep}(d,Q) &= \sum_{k=0}^{d-2} T\langle 0, k+1\rangle_c + (Q-d+1)\cdot T\langle 0, d\rangle_c \\ &\quad + \sum_{k=0}^{d-2} T\langle k+1, d-k-1\rangle_c + (d+Q-1)\cdot b. \end{aligned}$$

These expressions can be used to derive the optimal degree of pipelining (value of Q). The closed formulas for $T_{shallow}$ and T_{deep} can be found in the Appendix.

5 An Example: The Complete Exchange Problem

In this section, we consider the application of CALMANT to the Complete Exchange (or personalized All-to-All) communication problem. This problem is a global collective communication operation in which every process sends a unique block of data to every other process in the system. Since complete exchange arises in many important problems (i.e., FFT, matrix transposition, sorting), its efficient implementation on current parallel machines is an important research issue.

In order to apply CALMANT, we have designed a CC-cube algorithm for the Complete Exchange problem that is described in the next section. The resulting algorithm has been compared with a wide range of proposals specially tuned for the solution of the Complete Exchange: Binary [1], Quadrant [1], Modified Quadrant [14], Store-and-Forward (SAF) [14], Direct [11], Cyclic [12], and Hybrid [13] methods. Some of these proposals are addressed to 2-dimensional meshes and some others to c-dimensional meshes. The comparison is based on analytical models for the communication time and the results show that CALMANT outperforms the previous proposals in many cases.

5.1 A CC-cube Algorithm for Complete Exchange

In this section, we describe the CC-cube algorithm for complete exchange which is used as a starting point for the proposed methodology. Fig. 8 shows a particular example for $d = 3$.

Every node initially stores 2^d blocks of data in a vector of blocks denoted by M. Each block of data will be identified by the pair (n, j), where n is the source node and j is the

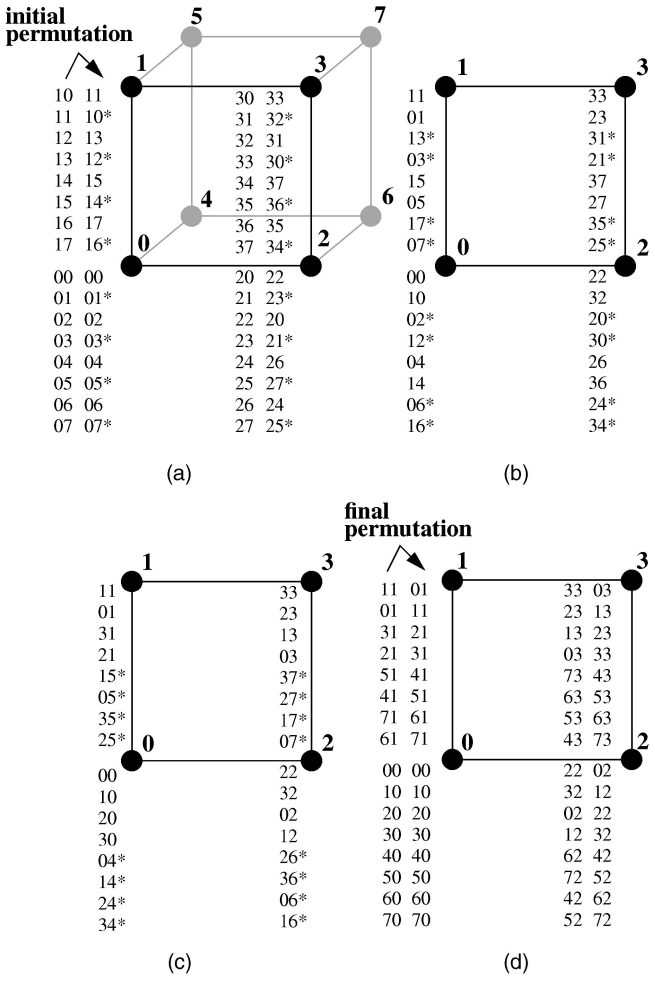

Fig. 8. A CC-cube algorithm for complete exchange for $d = 3$. (a) Contents of vectors M after the initial permutation. (b) Contents of M after exchange in dimension 0. (c) Contents of M after exchange in dimension 1. (d) Contents of M after exchange in dimension 2 and final permutation. Blocks to be sent in every iteration are marked with "*".

destination node for the corresponding block. Every node initially stores its 2^d blocks in a local vector of blocks denoted by M, in such a way that node n stores block (n, j) in position $M[j]$. For clarity, Fig. 8 only shows the blocks of data corresponding to nodes 0, 1, 2, and 3 of the CC-cube.

Initially, every node performs a permutation of vector M. After this permutation, block (n, j) is stored in $M[n\text{XOR}j]$, in node n (see Fig. 8a), where XOR denotes the bit-wise exclusive OR. As a result of this permutation, every node stores in $M[j]$ the block that, in order to reach its destination, must traverse those CC-cube dimensions that correspond to the bits which are set in the binary representation of j (i.e., $M[3]$ of node 1 contains the block $(1, 2)$ since this block must traverse dimensions 0 and 1 to reach its destination). Then, in every iteration i of the CC-cube, a subset of 2^{d-1} blocks are extracted from M to build the vector x_i that is sent to the neighbor along dimension i. In Fig. 8, the blocks which are selected in every iteration are marked with an asterisk. Because of the initial permutation, all the nodes obtain their corresponding blocks from the same locations of M. In particular, in iteration i a block in position $M[j]$ is selected if the ith bit of the binary representation of index j is set. After exchanging messages, the received blocks are stored in the locations of M that were occupied by the sent blocks.

After the three iterations required for this particular example, a new permutation is applied to leave the blocks in their final locations in M. This permutation is exactly the same as the initial one (see Fig. 8d).

The above algorithm was proposed by Johnsson and Ho [7] in order to minimize the time that blocks take since they leave the source nodes until they reach the destination nodes. We propose a slight modification of the algorithm in order to meet the requirements of the communication pipelining technique. This modification refers to the order in which the blocks are sent in each iteration. In particular, to build a message x_i, the blocks are always arranged in reverse order with regard to their positions in M. For instance, x_0 in node 0 contains blocks 07, 05, 03, and 01, in this order.

Although not shown in this paper, it can be proven that the requirements for communication pipelining are satisfied for all the cases.

5.2 Performance Figures

CALMANT has been compared with many other methods specially tuned for the Complete Exchange problem (they are enumerated at the beginning of Section 5). Nevertheless, for simplicity, we only show performance figures for those algorithms that outperform all the others for some scenarios. In other words, the methods not depicted in the figures are completely outperformed by those that are depicted.

Most of these proposals do not take into account the cost of barrier synchronization or the cost of copying blocks of data into buffers, so in order to compare the different algorithms, we assume both components are negligible. In our case, the cost of barrier synchronization is very small with respect to the total execution time; less than 1.6 percent with $\alpha/\tau = 500$, $b/\tau = 100$, and $B \geq 16$ in any of the considered scenarios, and the cost of copying blocks of data into buffers is null if $L/Q \leq B$.

We do not include in the performance study the ordering permutations. We just evaluate the communication cost of the algorithm. Depending on the problem, original data could be generated in the required order and consumed in the resulting order without extra permutations.

Fig. 9 and Fig. 10 plot the performance of different algorithms for several representative scenarios based on the analytical models described in previous sections. For every algorithm, we plot the speed-up over a baseline algorithm. This baseline algorithm corresponds to the original CC-cube. In the x-axis, we plot a logarithmic scale of the block size B. We consider a wide range of values for the machine parameters: number of nodes (2^d), dimensionality (c), and ratio α/τ. We have chosen ratios $\alpha/\tau =$ 100, 500, 1,000, and 5,000 as a representation of current computers (not necessarily with a mesh interconnection network): Cray T3D (567), IBM SP2 (1400), and Intel Paragon (5250).

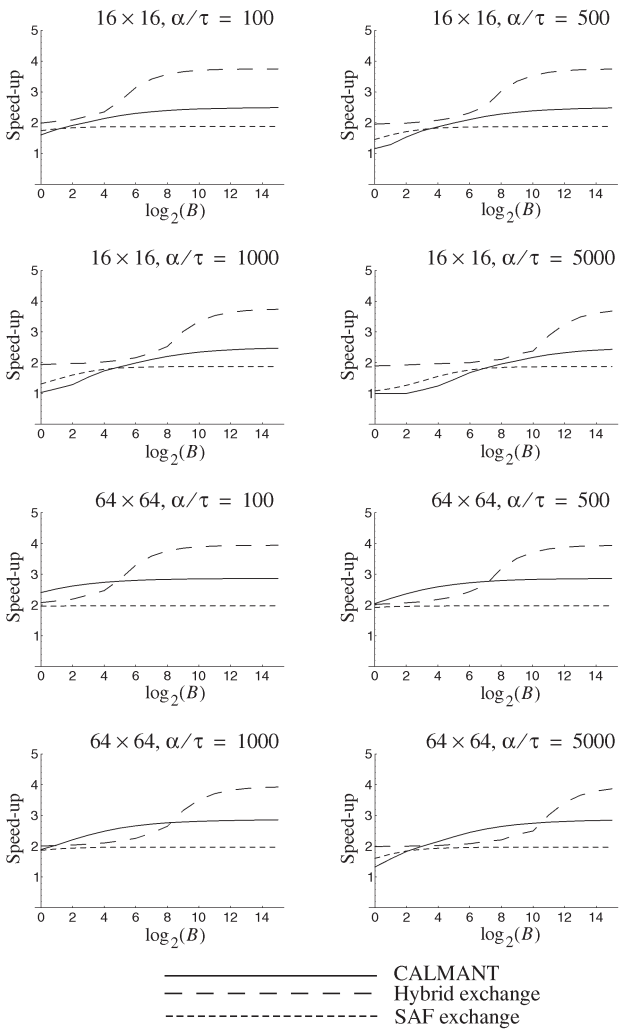

Fig. 9. Performance figures for 16×16 and 64×64 meshes.

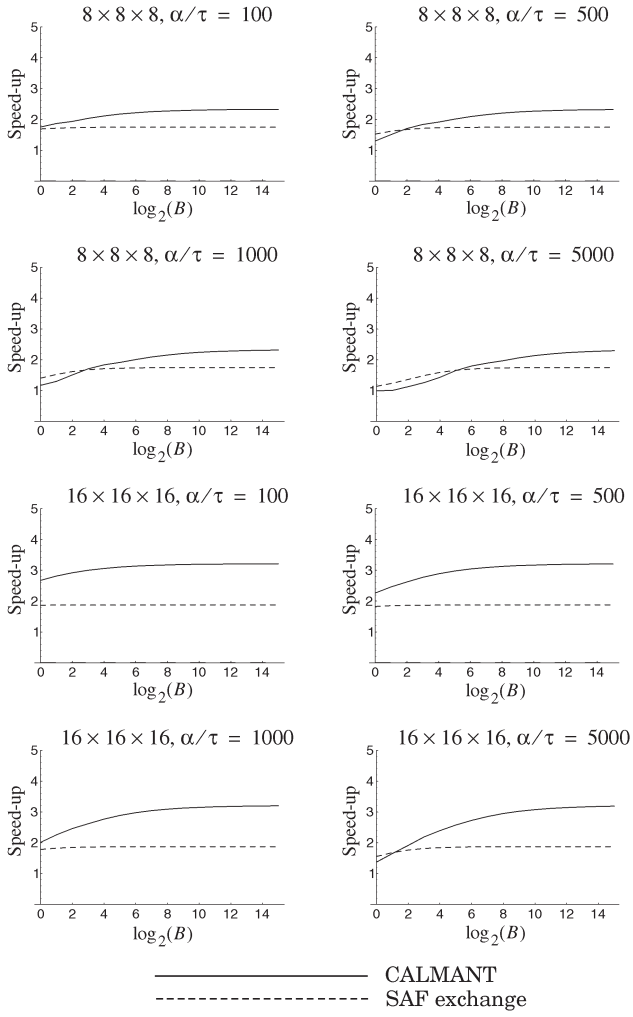

Fig. 10. Performance figures for $8 \times 8 \times 8$ and $16 \times 16 \times 16$ meshes.

Two important conclusions can be drawn from the plots:

- The proposed algorithm outperforms previous proposals for a significant range of values of the machine parameters and block size, and for many of the considered scenarios. For 2D mesh scenarios, the proposed algorithm outperforms the others for large values of d, large values of α/τ, and small values of B. The same holds for 3D mesh scenarios, but now for any value of B.
- In some cases ($16 \times 16 \times 16$ meshes), the proposed algorithm is about twice as fast as the best previous proposal.

6 Concluding Remarks

A systematic methodology (CALMANT) for mapping a certain type of algorithm on multicomputers with a mesh interconnection network has been proposed. The algorithms on which CALMANT focuses are called CC-cube algorithms. The recursive nature of CC-cube algorithms is very suitable to solve many computational problems such as the Fast Fourier Transform, the Complete Exchange communication problem (also known as All-to-All personalized communication) or the Jacobi method for single value decomposition and eigenvalue computation.

CALMANT is based on three different techniques: embedding of hypercubes on meshes, communication pipelining, and efficient message-scheduling algorithms. We have used the standard embedding since it is optimal in the sense of reducing the average dilation under the constant dilation property. The communication pipelining technique is inspired in the software pipelining approach used to generate code for VLIW processors. These two techniques are not new, but the combination of both results in an original methodology that allows us to map CC-cube algorithms in mesh interconnected computers in a systematic and efficient way. We have developed message-scheduling algorithms that minimize the number of communication steps required in the different communication stages of the CC-cube for one-port lines. Then, these message-scheduling algorithms have been extended to one-port multidimensional meshes. In this case, the proposed message-scheduling algorithms are also optimal, except in a very few particular situations, where they are very close to the optimal. In both cases, for line and meshes, we have presented analytical models of the execution time of the algorithms.

The Complete Exchange communication problem has been chosen as an example of application. A comparison between several proposals specially tuned to solve this particular problem on meshes and the algorithm that results from CALMANT reveals that the latter outperforms previous proposals for many machine configurations.

In this paper, we have only considered the case of one-port meshes, but CALMANT can be extended to all-port meshes and multiprocessor systems with a torus interconnection network. The most important conclusion is that we have proposed a systematic methodology that can be applied to a wide range of algorithms and architectures, and produces efficient parallel algorithms.

APPENDIX A

This appendix includes Theorem 1 and Theorem 3 and their corresponding proofs. Both theorems are related to the maximum load for lines and meshes, respectively.

A.1 Maximum Load for Lines

A task $\langle i, M\rangle_1$ can be decomposed into a set of M subtasks $\{\langle k,1\rangle_1 | k \in [i, i+M-1]\}$ and the load of any link can be evaluated as the sum of the load due to each of these subtasks. First, we will focus on a particular subtask $\langle k,1\rangle_1$ and then we will evaluate the maximum load due to the entire set of subtasks.

Let n be a process of the pipelined CC-cube; let n' be the neighbor process n along dimension k (i.e., $n' = N_k(n)$) and let p be the node such that $p = f_{std}(n)$. We refer to the node p' such that $p' = f_{std}(n')$ as the *partner* node of p along dimension k. Note that, according to the definition of the standard embedding, it results that $p = n$ and $p' = n'$. When the subtask $\langle k,1\rangle_1$ is performed, the process n sends and receives a message to/from the process n'. The distance between the node p and its partner p' is D_k where, according to expression (2) in Section 2.3:

$$D_k = 2^k.$$

If p' is one of the nodes located on the right-hand (left-hand) side of node p, the node p sends a message that travels from left to right (right to left) direction and receives a message that travels in opposite direction. This can be extended to all the nodes of the line and, therefore, the number of messages that traverse any link in one direction is equivalent to the number of messages that traverse the same link in the opposite direction. According to the definition of load of a link, we can just consider one direction since the load is the same for both. Without loss of generality, we will consider only messages that travel from left to right. The following lemma states the location of node p' in relation to node p.

Lemma 3. *If p is a node of the line such that $p \bmod 2^{k+1} < 2^k$ ($2^k \le p \bmod 2^{k+1} < 2^{k+1}$) and p' is its partner node along dimension k, then p' is located at the right (left) side of p at a distance 2^k.*

Proof. Let $\langle n_{d-1}, n_{d-2}, ..., n_0\rangle$ be the binary representation of n and let $\langle p_{d-1}, p_{d-2}, ..., p_0\rangle$ be the binary representation of p, such that $p = f_{std}(n)$. According to the definition of the standard embedding:

$$p_l = n_l; \forall l \in [0, d-1].$$

Let $\langle n'_{d-1}, n'_{d-2}, ..., n'_0\rangle$ be the binary representation of n' and let $\langle p'_{d-1}, p'_{d-2}, ..., p'_0\rangle$ be the binary representation of p', such that $p' = f_{std}(n')$. If $n' = N_k(n)$ then we have that:

$$n'_l = \begin{cases} n_l, & \text{if } l \neq k; \\ \overline{n_l}, & \text{if } l = k, \end{cases}$$

and therefore:

$$p'_l = \begin{cases} n_l = p_l, & \text{if } l \neq k; \\ \overline{n_l} = \overline{p_l}, & \text{if } l = k. \end{cases} \quad (3)$$

If $p \mod 2^{k+1} < 2^k$, then $p_k = 0$ and according to (3):

$$p'_l = \begin{cases} p_l, & \text{if } l \neq k; \\ 1, & \text{if } l = k, \end{cases}$$

so the partner of p is located on its right-hand side. If $2^k \le p \mod 2^{k+1} < 2^{k+1}$, then $p_k = 1$ and according to (3):

$$p'_l = \begin{cases} p_l, & \text{if } l \neq k; \\ 0, & \text{if } l = k, \end{cases}$$

so the partner of p is located on its left-hand side. □

Next, the load of the links due to a subtask $\langle k,1\rangle_1$ is evaluated. Let $\lambda_k(m)$, such that $m \in [0, 2^{d-2}]$ and $k \in [0, d-1]$, be the load of the link that is located between node m and node $m+1$ due to the subtask $\langle k,1\rangle_1$. The following lemma gives an expression for $\lambda_k(m)$.

Lemma 4. *The load of the link between node m and node $m+1$ due to subtask $\langle k,1\rangle_1$ is given by the following expression:*

$$\lambda_k(m) = \begin{cases} (m \bmod 2^k) + 1, & \text{if } m \bmod 2^{k+1} < 2^k; \\ 2^k - (m \bmod 2^k) - 1, & \text{if } 2^k \le m \bmod 2^{k+1} < 2^{k+1}. \end{cases}$$

Proof. If a message traverses the link between node m and node $m+1$ in the right direction and p is the source node of that message, given that 2^k is the distance between node p and its partner along dimension k, we necessarily have that $p \in [m - 2^k + 1, m]$. For the rest of this proof, we will only consider this set of nodes, since all the source nodes of the messages that traverse the link between nodes m and $m+1$ in right direction when the subtask $\langle k,1\rangle_1$ is performed belong to this set.

If $m \bmod 2^{k+1} < 2^k$ ($2^k \le m \bmod 2^{k+1} < 2^{k+1}$) for any node p such that $p \in [m - 2^k + 1, m - (m \bmod 2^k) - 1]$, we have that $2^k \le p \bmod 2^{k+1} < 2^{k+1}$ ($p \bmod 2^{k+1} < 2^k$) and according to Lemma 3, the partner of p is located on its left-hand (right-hand) side. Otherwise, for any node p such that $p \in [m - (m \bmod 2^k), m]$, we have that $p \bmod 2^{k+1} < 2^k$ ($2^k \le p \bmod 2^{k+1} < 2^{k+1}$) and according to Lemma 3, the partner of p is located on its right-hand (left-hand) side. Therefore, the total number of messages that traverse the link between nodes m and $m+1$ in the right direction is $(m \bmod 2^k) + 1$ ($2^k - (m \bmod 2^k) - 1$). □

So far, we have evaluated the load of the links due to one of the subtasks $\langle k,1\rangle_1$ in which the task $\langle i, M\rangle_1$ is decomposed. The study of a subtask $\langle k,1\rangle_1$ has been the

first step to evaluate the load due to the task $\langle i, M\rangle_1$. The following theorem gives an expression for the maximum load of a line due to a task $\langle i, M\rangle_1$.

Theorem 1. *The maximum load of a line when a task $\langle i, M\rangle_1$ is performed can be computed as follows:*

$$\lambda_{max}\langle i, M\rangle_1 = \begin{cases} (2^{i+M+1} - 2^{i+1}/3, & \text{if } M \text{ is even;} \\ (2^{i+M+1} - 2^i/3, & \text{if } M \text{ is odd.} \end{cases}$$

Proof. The maximum load of a line due to a subtask $\langle k, 1\rangle_1$ is given by Lemma 4. Using that expression, which gives the load of the link between nodes m and $m+1$, we have that:

$$\lambda_{max}\langle k, 1\rangle_1 = \lambda_k(m) = 2^k; \forall\{m | m \mod 2^{k+1} = 2^k - 1\}.$$

The load of the link between node m and node $m+1$ due to subtask $\langle k, 2\rangle_1$ (which comprises both subtasks $\langle k, 1\rangle_1$ and $\langle k+1, 1\rangle_1$) can be evaluated as $\lambda_k(m) + \lambda_{k+1}(m)$. Applying the expression given by Lemma 4, we have that:

$$\lambda_k(m) + \lambda_{k+1}(m) = \begin{cases} f_1(m,k), & \text{if } 0 \le m \mod 2^{k+1} < 2^k; \\ 2^{k+1}, & \text{if } 2^k \le m \mod 2^{k+1} < 2^{k+1} + 2^k; \\ f_2(m,k), & \text{if } 2^{k+1} + 2^k \le m \mod 2^{k+1} < 2^{k+2}, \end{cases}$$

where $f_1(m,k), f_2(m,k) \le 2^{k+1}$. Thus, the maximum load is $\lambda_{max}\langle k, 2\rangle_1 = 2^{k+1}$ and:

$$\begin{aligned} &\lambda_{max}\langle k, 2\rangle_1 = \lambda_k(m) + \lambda_{k+1}(m) = 2^{k+1}; \\ &\forall\{m | 2^k \le m \mod 2^{k+1} < 2^{k+1} + 2^k\}. \end{aligned} \tag{4}$$

The expression $\lambda_k(m) + \lambda_{k+1}(m)$ is a periodic function with a period $4 \cdot 2^k$. Moreover, we can find at least one interval of size $2 \cdot 2^k$ in which the expression is constant and achieves its maximum value. According to Lemma 4, the expression $\lambda_l(m)$ is a periodic function with a period 2^{l+1} and, therefore, the following expression:

$$\sum_{l=a}^{k-1} \lambda_l(m); \forall a \in [0, k-1] \tag{5}$$

is also a periodic function and its period is 2^k.

We can find at least one complete period of (5) such that for any m belonging to that period, the expression $\lambda_k(m) + \lambda_{k+1}(m)$ is constant and achieves its maximum value. In this way, using (4) given above we have that:

$$\begin{aligned} &\max_{\forall m}\left\{\sum_{l=a}^{k+1} \lambda_l(m)\right\} = \lambda_{max}\langle k, 2\rangle_1 \\ &+ \max_{\forall m}\left\{\sum_{l=a}^{k-1} \lambda_l(m)\right\}; \forall a \in [0, k-1]. \end{aligned} \tag{6}$$

The maximum load due to task $\langle i, M\rangle_1$ can be computed as follows:

$$\lambda_{max}\langle i, M\rangle_1 = \max_{\forall m}\left\{\sum_{k=0}^{M-1} \lambda_{i+k}(m)\right\},$$

and, therefore, recursively applying (6) we obtain:

$$\lambda_{max}\langle i, M\rangle_1 = \begin{cases} \sum_{k=0}^{M/2-1} \lambda_{max}\langle i+2k, 2\rangle_1, & \text{if } M \text{ is even,} \\ \lambda_{max}\langle i, 1\rangle_1 + \sum_{k=1}^{(M-1)/2} \lambda_{max}\langle i+2k-1, 2\rangle_1, & \text{if } M \text{ is odd.} \end{cases}$$

□

A.2 Maximum Load for Meshes

The following lemma and theorem establishes a relationship between the maximum load of a c-dimensional mesh and the maximum load of a line.

Lemma 5. *The maximum load of a c-dimensional mesh due to a communication task in which the nodes exchange messages through the dimensions $\{i, i+c, ..., i+(M-1)\cdot c\}$ of a CC-cube, is equivalent to the maximum load of a line due to the communication task $\langle \lfloor i/c \rfloor, M\rangle_1$.*

Proof. Because of the standard embedding definition, the dimensions of a CC-cube are mapped cyclically on the axes of the mesh. Therefore, all the dimensions $\{i, i+c, ..., i+(M-1)\cdot c\}$ are mapped onto the axis $i \bmod c$. The dilations of these dimensions (see (2) in Section 2.3) are $2^{\lfloor i/c \rfloor}, 2^{\lfloor i/c \rfloor + 1}, \ldots, 2^{\lfloor i/c \rfloor + M - 1}$, respectively, which are just the same dilations as those involved in the task $\langle \lfloor i/c \rfloor, M\rangle_1$. □

Theorem 3. *The maximum load due to the communication task $\langle i, M\rangle_c$ on a $(2^{d/c} \times ... \times 2^{d/c})$ c-dimensional mesh is equivalent to the maximum load due to the task $\langle \lfloor (i + (M-1) \bmod c)/c \rfloor, \lceil M/c \rceil\rangle_1$ on a line.*

Proof. The set of dimensions involved in a task $\langle i, M\rangle_c$ can be decomposed into c separate subsets, so that the dimensions involved in each subset are mapped onto the same axis of the mesh. The dimensions involved in each one of these subsets are:

$$\begin{aligned} S(j) = \{&i+j, i+j+c, \ldots, i+j \\ &+ (\lceil (M-j)/c \rceil - 1)\cdot c\}; \forall j \in [0, c-1]. \end{aligned}$$

By Lemma 5, we know that the maximum load due to any of the above subsets is:

$$\lambda_{max}\langle \lfloor (i+j)/c \rfloor, \lceil (M-j)/c \rceil\rangle_1.$$

The highest maximum load is that in which the dimensions with highest dilations are involved, that is to say, the subset $S((M-1) \bmod c)$. By simple substitution, we can conclude that:

$$\lambda_{max}\langle i, M\rangle_c = \lambda_{max}\langle \lfloor (i + (M-1) \bmod c)/c \rfloor, \lceil M/c \rceil\rangle_1.$$

□

A.3 Closed Formulas of Analytical Models

This section contains the closed formulas for $T_{shallow}$ and T_{deep} expressions. The final formulas are the result of a mathematical process that is not included in this work. Note that $T_{shallow}$ and T_{deep} expressions are equivalent when $Q = d$.

On a Line

$$T_{shallow}(d,Q) = \frac{(3Q+1)2^{d+2} + f_1(d,Q)}{18}\left(\alpha + \frac{N}{Q}\tau\right) + (d+Q-1)b,$$

where:

$$f_1(d,Q) = \begin{cases} -2^{d-Q+2} - 9Q & \text{if } Q \text{ is even;} \\ 2^{d-Q+2} - 9Q - 9 & \text{if } Q \text{ is odd.} \end{cases}$$

$$T_{deep}(d,Q) = \frac{(3Q+1)2^{d+2} + f_2(d,Q)}{18}\left(\alpha + \frac{N}{Q}\tau\right) + (d+Q-1)b,$$

where:

$$f_2(d,Q) = \begin{cases} 3d - 12Q - 4 & \text{if } d \text{ is even;} \\ -3d - 6Q - 5 & \text{if } d \text{ is odd.} \end{cases}$$

On a c-dimensional Mesh

$$T_{shallow}(d,Q) = \frac{(c+3c\lceil Q/c\rceil + 3r)2^{d/c+2} + f_3(d,Q)}{18}\left(\alpha + \frac{N}{Q}\tau\right) + (d+Q-1)b,$$

where:

$$f_3(d,Q) = \begin{cases} -(c+3r)2^{d/c-\lceil Q/c\rceil+2} - 9c\lceil Q/c\rceil & \text{if } \lceil Q/c\rceil \text{ is even;} \\ (2c-3r)2^{d/c-\lceil Q/c\rceil+1} - 9c(\lceil Q/c\rceil + 1) & \text{if } \lceil Q/c\rceil \text{ is odd,} \end{cases}$$

and $r = (Q-1) \bmod c$.

$$T_{deep}(d,Q) = \frac{(3Q+4c-3)2^{d/c+2} + f_4(d,Q)}{18}\left(\alpha + \frac{N}{Q}\tau\right) + (d+Q-1)b,$$

where:

$$f_4(d,Q) = \begin{cases} 3d - 12Q - 16c + 12, & \text{if } d/c \text{ is even;} \\ -3d - 6Q - 11c + 6, & \text{if } d/c \text{ is odd.} \end{cases}$$

ACKNOWLEDGMENTS

This work has been supported by the Ministry of Education and Science of Spain and the European Union (CICYT TIC-98/0511 and CICYT TIC-2001/0995-C02-01) and the European Center for Parallelism of Barcelona (CEPBA).

REFERENCES

[1] S.H. Bokhari and H. Berryman, "Complete Exchange on a Circuit Switched Mesh," *Proc. IEEE Scalable High Performance Computing Conf.*, pp. 300-306, 1992.

[2] L.M. Díaz de Cerio, A. González, and M. Valero-García, "Communication Pipelining in Hypercubes," *Parallel Processing Letters*, vol. 6, no. 4, pp. 507-523, 1996.

[3] L.M. Díaz de Cerio, M. Valero-García, and A. González, "A Method For Exploiting Communication/Computation Overlap in Hypercubes," *Parallel Computing*, vol. 24, no. 2, pp. 221-245, 1998.

[4] A. González and M. Valero-García, "The Xor Embedding: An Embedding of Hypercubes onto Rings and Toruses," *Proc. Int'l Conf. Applications Specific Array Processors*, pp. 15-28, 1993.

[5] A. González, M. Valero-García, and L.M. Díaz de Cerio, "Executing Algorithms with Hypercube Topology on Torus Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 8, pp. 803-814, 1995.

[6] S.L. Johnsson and R.L. Krawitz, "Cooley-Tukey FFT on the Connection Machine," *Parallel Computing*, no. 18, pp. 1201-1221, 1992.

[7] S.L. Johnsson and C.T. Ho, "Optimal All-to-All Personalized Communication with Minimum Span on Boolean Cubes," *Proc. Sixth IEEE Distributed Memory Computing Conf.*, pp. 299-304, 1991.

[8] T.H. Lai and A.P. Sprague, "Placement of the Processors of a Hypercube," *IEEE Trans. Computers*, vol. 40, no 6, pp. 714-722, June 1991.

[9] M. Lam, "Software Pipelining: An Effective Schedulling Technique for VLIW Machines," *Proc. Conf. Programming Language Design and Implementation*, pp. 318-328, 1988.

[10] S. Matic, "Emulation of Hypercube Architecture on Nearest-Neighbor Mesh-Connected Processing Elements," *IEEE Trans. Computers*, vol. 39, no. 5, pp. 698-700, May 1990.

[11] D.S. Scott, "Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies," *Proc. Sixth IEEE Distributed Memory Computing Conf.*, pp. 398-403, 1991.

[12] N.S. Sundar, D.N. Jayasimha, D.K. Panda, and P. Sadayappan, "Complete Exchange in 2D Meshes," *Proc. IEEE Scalable High Performance Computing Conf.*, pp. 406-413, 1994.

[13] N.S. Sundar, D.N. Jayasimha, D.K. Panda, and P. Sadayappan, "Hybrid Algorithms for Complete Exchange in 2D Meshes," *Proc. ACM Int'l Conf. Supercomputing*, pp. 181-188, 1996.

[14] S. Takkella and S. Seidel, "Complete Exchange and Broadcast Algorithms for Meshes," *Proc. IEEE Scalable High Performance Computing Conf.*, pp. 422-428, 1994.

Luis M. Díaz de Cerio received the degree in telecommunications engineering in 1993 and the PhD degree in telecommunication engineering in 1998, both from the Universitat Politcnica de Catalunya (UPC) at Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at the UPC. His research interests focus on parallel algorithms for multiprocessor systems, distributed computing systems, and Grid.

Miguel Valero-García received the degree in computer science in 1986 and the PhD degree in computer science in 1989, both from the Universitat Politcnica de Catalunya (UPC) at Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at the UPC. His research interests focus on parallel algorithms for multiprocessor systems and new techniques to improve the academic quality at university.

Antonio González received the engineering degree in computer science in 1986 and the PhD degree in computer science in 1989, both from the Universitat Politcnica de Catalunya (UPC), Barcelona, Spain. He has occupied different faculty positions in the Computer Architecture Department at the UPC since 1986, with tenure since 1990. His research interests center on computer architecture, compilers, and parallel processing, with a special emphasis on processor microarchitecture, memory hierarchy, and instruction schedulling. Dr. Gonzalez is a member of the IEEE Computer Society.